Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gaber Žinko

Razvoj sistema za rezervacijo termina v frizerskem salonu

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana, 2016

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Razvoj sistema za rezervacijo strank v frizerskem salonu

Tematika naloge:

Za podjetja kot so na primer frizerski saloni je pomembno, da je stranka zadovoljna z uslugo. Eden izmed dejavnikov pri zadovoljstvu strank je tudi, da stranka ne čaka predolgo na storitev. V ta namen razvijte sistem, ki bo stranki omogočal enostavno izbiro in rezervacijo želenega termina v frizerskem salonu ter potrditev želenega termina s strani izbranega frizerja. Sistem naj poleg tega omogoča tudi predstavitev frizerskega salona. Pri razvoju sistema izberite primerne tehnologije na strani strežnika in na strani odjemalca. Sistem zasnujte tako, da bo odjemalski del deloval na izbrani mobilni platformi.

Iskreno se zahvaljujem svoji družini, prijateljem in vsem, ki so mi nudili podporo in pomoč v času študija. Posebej se pa zahvaljujem doc. dr. Alešu Smrdelu, za strokovno pomoč in nasvete pri pisanju diplomskega dela.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Cilji diplomske naloge 1

1.2 Pregled področja 2

1.3 Struktura diplomske naloge 2

2 Uporabljene tehnologije in orodja **3**

2.1 Tehnologije 3

2.2 Knjižnice in ogrodja 7

2.3 Orodja 12

3 Podatkovni model **15**

3.1 Koraki razvoja podatkovnega modela 15

3.2 Opis modela 17

3.3 Shema podatkovne baze 19

4 Razvoj spletne aplikacije **21**

4.1 Uporabniški vmesnik 23

4.2 ASP.NET MVC 23

4.3 Web API 26

4.4 Varnost 27

5 Razvoj Android aplikacije **29**
5.1 Aplikacija namenjena strankam 31
5.2 Aplikacija namenjena zaposlenim 33
5.3 Zasnova uporabniškega vmesnika 34
5.4 Varnost 35

6 Predstavitev uporabe **37**
6.1 Stranke 38
6.2 Frizer 44

7 Sklepne ugotovitve in nadaljnje delo **47**
7.1 Nadaljnje delo 48

Literatura **51**

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application programming interface	Aplikacijski programski vmesnik
LINQ	Language integrated query	Jezikovna integrirana poizvedba
HTML	Hypertext markup language	Jezik za označevanje nadbesedila
CSS	Cascading style sheets	Kaskadne slogovne podloge
REST	Representational state transfer	Prenos predstavitvenega stanja
SOAP	Simple object access protocol	Preprost dostopni protokol za objekte
HTTP	Hypertext transfer protocol	Protokol za prenos nadbesedila
URL	Uniform resource locator	Enolični krajevnik vira
XML	Extensible markup language	Razširljivi označevalni jezik
JSON	JavaScript object notation	JavaScript notacija objekta
DOM	Document object model	Objektni model dokumenta
CLR	Common language runtime	Skupno izvajalno okolje programskih jezikov
CLI	Common language infrastructure	Skupna infrastruktura programskih jezikov
FCL	Framework class library	Knjižnica razredov ogrodja
UWP	Universal Windows platform	Univerzalna Windows platforma
MVC	Model-view-controller	Model-pogled-nadzornik
SQL	Structured query language	Strukturirani povpraševalni jezik
IDE	Integrated development environment	Integrirano razvojno okolje
ADT	Android development tools	Razvojna orodja za Android
SDK	Software development kit	Pribor za razvoj programske opreme
CRM	Customer relationship management	Upravljanje odnosov s strankami

Povzetek

Naslov: Razvoj sistema za naročanje strank v frizerskem salonu

Razvit je bil sistem, ki omogoča uporabniku hitro in preprosto rezervacijo termina pri želenem frizerju v frizerskem salonu. Sistem je sestavljen iz spletne aplikacije in Android aplikacije. Spletna aplikacija omogoča preprosto administracijo, beleženje števila naročil posameznih strank, poleg tega pa vsebuje spletni aplikacijski programski vmesnik, ki služi kot vmesnik med podatkovno bazo in Android aplikacijo. Android aplikacija je namenjena tako strankam, kot tudi zaposlenim v frizerskem salonu. Stranka odda naročilo, ki se nato shrani v podatkovno bazo na strežniku, pri čemer izbran frizer dobi obvestilo na svoji Android aplikaciji in termin odobri ali pa vzpostavi direktno povezavo s stranko.

Ključne besede: frizerski salon, spletna storitev, Android, spletno naročanje.

Abstract

Title: English title

A system was developed, that allows users to quickly and easily book an appointment in a hair salon. The system consists of a web application and an Android application. The web application enables simple administration, logging the number of customer appointments, and in addition implements a web application programming interface, that serves as an interface between the database and the Android application. The Android application is designed for both customers as well as employees in the hair salon. The customer submits an order, which is then stored in the database on a server, wherein the selected hairdresser gets a notification on his or her Android application, and either confirms the selected appointment or establishes a direct link with the customer.

Keywords: hair salon, web service, Android, online booking.

Poglavje 1

Uvod

Pametni telefoni so danes že tako razširjeni, da si težko predstavljamo življenje brez njih. Njihov razvoj je ljudem omogočil popolnoma nove načine interakcije z drugimi ljudmi. Še posebej pa je tak pristop zanimiv za podjetja, ki iščejo nove načine komunikacije s svojimi strankami. Pri teh načinih komunikacije pa se lahko uporabljajo klasične spletne aplikacije, pogosto pa se v ta namen uporabljajo namenske mobilne aplikacije. Tovrstne aplikacije po funkcionalnosti segajo vse od preprostih informacijskih aplikacij, preko katalogov storitev, pa vse do velikih sistemov, ki že sami po sebi ponujajo razne storitve. V tem diplomskem delu bomo razvili sistem, ki omogoča uporabnikom rezervacijo termina v frizerskem salonu. Osredotočili se bomo na specifičen salon, a bo sistem sestavljen tako, da bo z nekaj spremembami prenosljiv na katerikoli drug frizerski salon ali še celo katerokoli drugo podjetje, ki išče nov način naročanja svojih strank.

1.1 Cilji diplomske naloge

Za cilj naše naloge smo si zadali razvoj sistema za rezervacijo terminov v frizerskem salonu. Strankam salona mora na preprost intuitiven način ponujati izbiro frizerja, storitve in termina. Ker je zaželjeno, da je sistem prenosljiv, smo se odločili za implementacijo odjemalskega dela na mobilnih napravah

z operacijskim sistemom Android. Poleg uporabniške aplikacije mora sistem za delovanje vsebovati tudi API, ki deluje na strežniku, ter preprosto spletno administracijsko konzolo, kjer ima administrator vpogled nad celotnim sistemom. Tudi ta konzola mora biti preprosta za uporabo, saj od administratorja ne moremo pričakovati znanja računalništva.

1.2 Pregled področja

Tovrstnih sistemov je na trgu že kar nekaj. Na primer podjetje Spletnik [1] ponuja podoben sistem v sklopu izdelave spletne strani. Večinoma pa se takšni sistemi uporabljajo za rezervacijo hotelskih sob, kot na primer podjetje Hotelinco [2], in je pravzaprav zelo redko videti takšen sistem v uporabi pri storitvenih podjetjih, kot so na primer frizerski saloni. A vsi ti sistemi so spletne aplikacije, ki na mobilnih napravah preko brskalnika sicer delujejo, a po našem mnenju mobilna aplikacija, razvita za neko mobilno platformo, v našem primeru je to sistem Android, ponuja veliko boljšo uporabniško izkušnjo.

1.3 Struktura diplomske naloge

V prvem poglavju predstavljamo temo diplome. V drugem poglavju predstavljamo tehnologije, knjižnice, orodja in ogrodja, ki smo jih uporabili pri razvoju. V tretjem poglavju predstavljamo podatkovni model našega sistema, ter opisujemo kako hranimo potrebne podatke in kakšna je zgradba naše podatkovne baze. Sledita dve poglavji, ki opisujeta celoten postopek izdelave spletne in Android aplikacije. Šesto poglavje pa opisuje Android aplikacijo z vidika stranke in frizerja salona. V njem predstavimo vse aktivnosti in funkcionalnosti naše aplikacije. V sedmem poglavju pa so navedeni zaključki in ideje za nadaljne delo.

[1] `https://spletnik.si/`
[2] `http://www.hotelinco.eu/`

Poglavje 2

Uporabljene tehnologije in orodja

V tem poglavju bomo predstavili nekaj tehnologij in orodij, s katerimi smo razvili naš sistem. V prvem delu bomo opisali tehnologije, ki smo jih uporabili. V drugem delu bomo opisali ogrodja in glavne knjižnice, ki so vključene v sistemu. Za konec pa bomo opisali še orodja oz. okolja v katerih smo naš sistem razvijali.

2.1 Tehnologije

Pod tehnologije uvrščamo razne programske ter označevalne jezike, kot tudi razne pristope k razvoju spletne in Android aplikacije ter vzpostavitve komunikacije med strežnikom in odjemalcem.

2.1.1 Java

Java je visokonivojski programski jezik, ki ga je leta 1995 izdalo podjetje Sun Microsystems, ki je sedaj del podjetja Oracle [1]. Prvotni namen je bil ustvariti jezik, katerega kodo napišemo enkrat in jo poganjamo povsod (ang. Write once run anywhere) [2]. Cilj je bil ustvariti programski jezik, ki je neodvisen od platforme, kjer se koda poganja. Sintaksa je bila osnovana

po zgledu programskega jezika C++, vendar je preprostejša in je primarno objektno usmerjen jezik. Danes se Java uporablja na velikem spektru področij, tako na spletu kot izven. V tem diplomskem delu je jezik uporabljen pri razvoju Android aplikacije.

2.1.2 C#

C# je programski jezik podjetja Microsoft, ki temelji na jeziku C++, a je veliko navdiha dobil od programskega jezika Java. Cilj je bil ustvariti preprost in moderen objektno usmerjen jezik, ki omogoča uporabo najboljših lastnosti predhodnih jezikov, kot so Java, C in C++, ter odpravlja nekaj njihovih pomanjkljivosti. C# cilja na izvajalsko okolje CLI in s tem ogrodje .NET. Zaradi tega je kodo C# mogoče enostavno povezovati z deli, ki so napisani v kateremkoli drugem programskem jeziku skladnem s CLI. Programi napisani v C# imajo tudi zelo dobro prenosljivost med različnimi platformami, pri čemer si moramo pomagati s prenosljivimi implementacijami CLI, kot sta na primer Mono ali .NET Core [3].
Za C# smo se odločili zato, ker nam omogoča delo v ogrodju .NET, ki že samo po sebi vsebuje veliko orodij ter ustaljenih praks pri razvoju spletnih aplikacij in storitev. Kar se tiče sintakse je zelo podoben jeziku Java, a je na določenih področjih precej bolj napreden. Med drugim podpira na primer dinamične spremenljivke, vsebuje LINQ in ima bolj dodelane generike.

2.1.3 HTML

HTML je označevalni jezik, namenjen izdelavi spletnih strani [4]. Prvič je bil javno opisan leta 1991 in je hitro postal standard na svojem področju. Določa osnovno zgradbo spletnih dokumentov (slika 2.1) in omogoča njihov prikaz s pomočjo spletnega brskalnika. HTML kodo lahko pišemo v kateremkoli urejevalniku besedil in jo od navadnega teksta ločimo z značkami. Ker je jezik standardiziran, je dokumente HTML mogoče prikazati v kateremkoli brskalniku. Najnovejša različica HTML5 je bila izdana leta 2014 in je

poudarila integracijo različnih multimedijskih vsebin.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title></title>
</head>
<body>

</body>
</html>
```

Slika 2.1: Osnovna struktura praznega dokumenta HTML

2.1.4 CSS

CSS (kaskadne slogovne podloge) so preprost slogovni jezik, ki skrbi za vizualno predstavitev spletnih strani [5]. Koda je lahko vgrajena v sam dokument HTML, a je pogosto od njega ločena in zapisana v svoji lastni datoteki s končnico .css. Pri izdelavi spletne strani ta ločenost omogoča večjo fleksibilnost in boljši pregled nad izdelavo strani, obenem pa omogoča enostavno zagotovitev uniformnega izgleda večih strani, ki uporabljajo določene podloge. S podlogami, kot je prikazano na sliki 2.2, urejamo celoten izgled spletnih dokumentov (barve, pisava, pozicije elementov itd.), nimajo pa vpliva na samo strukturo.

```
body {
    padding-top: 50px;
    padding-bottom: 20px;
}
```

Slika 2.2: Primer določanja lastnosti elementom z uporabo CSS

2.1.5 JavaScript

JavaScript je objektno usmerjen skriptni jezik. Namenjen je ustvarjanju interaktivnih spletnih strani. Kljub imenu JavaScript ni vezan na programski jezik Java, a si z njim deli mnoge lastnosti [6]. Podobno kot CSS je lahko

koda JavaScript vgrajena kar v sam dokument HTML, a jo je zaradi boljše preglednosti ponavadi bolje zapisati v samostojno datoteko s končnico .js in nato dodati v dokumentu HTML le klic na to datoteko.

2.1.6 Razor

Razor je označevalna sintaksa, ki omogoča integracijo strežniške kode (Visual Basic ali C#) v spletni dokument HTML (slika 2.3). Razor omogoča dinamičen način ustvarjanja vsebine. Temelji na ASP.NET in je namenjen ustvarjanju spletnih aplikacij [7]. Sintaksa je podobna sintaksi skriptnega jezika PHP. Ena njegovih najpomembnejših lastnosti je zmožnost neposredne komunikacije s podatkovnimi bazami.

```
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li><a asp-controller="Home" asp-action="Index">Home</a></li>
        <li><a asp-controller="Frizers" asp-action="Index">Frizerji</a></li>
    </ul>
    @await Html.PartialAsync("_LoginPartial")
</div>
```

Slika 2.3: Primer klicev funkcij .NET vgrajenih v kodo HTML

2.1.7 REST

REST je pristop h komunikaciji med strežnikom in odjemalcem, ki je pogosto uporabljen pri razvoju spletnih storitev. V nasprotju s protokolom SOAP je REST le arhitekturno načelo, ki kot tako ni standardizirano [8]. Za manipulacijo s podatki uporablja protokol HTTP, pri čemer se uporabljajo preproste metode GET, PUT, POST in DELETE, ki so predstavljene v tabeli 2.1. Vsak vir v bazi je pri REST predstavljen kot enoznačni naslov URL. Ker REST sam po sebi ne določa tipa poslanih in prejetih podatkov, imamo lahko v nekem primeru dve metodi, ki vračata iste podatke, pri čemer lahko ena metoda vrne pogled z vstavljenimi podatki, ena pa preprost objekt JSON, ki ga lahko potem na primer brez problemov preberemo na odjemalčevi strani v aplikaciji Android.

Tabela 2.1: Prikaz metod pristopa REST

Storitev	**URL**	**Metoda**	**Parameter**
Prenos vseh podatkov	http://streznik/stranke	GET	/
Prenos enega podatka	http://streznik/stranke/5	GET	ID podatka
Posodabljanje podatka	http://streznik/stranke/5	PUT	ID podatka
Dodajanje podatka	http://streznik/stranke	POST	/
Brisanje vseh podatkov	http://streznik/stranke	DELETE	/
Brisanje enega podatka	http://streznik/stranke/5	DELETE	ID podatka

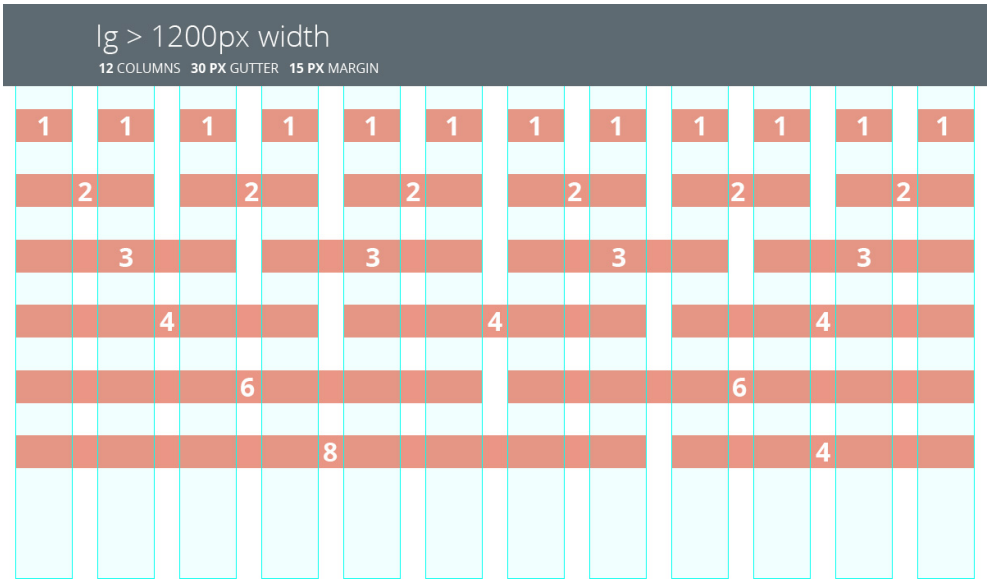
2.2 Knjižnice in ogrodja

V tem predelu bomo predstavili knjižnice in ogrodja, ki smo jih uporabili pri razvoju sistema. Knjižnice so zbirke funkcij, namenjene pomoči pri razvoju programske opreme. Ponavadi so napisane na takšen način, da se jih da na preprost način uporabiti v kateremkoli programu. Ogrodja po drugi strani pa so v primerjavi s knjižnicami mnogo bolj obsežna. Sem spadajo Bootstrap, jQuery, pa tudi večja ogrodja kot je na primer .NET, ki imajo podporo velikih podjetij, ki veliko vlagajo v njihov razvoj in uporabnost.

2.2.1 Bootstrap

Bootstrap je odprtokodno ogrodje za oblikovanje spletnih strani in aplikacij. Vsebuje predloge HTML in CSS, ki določajo tipografijo, gumbe, navigacijo ter postavitev elementov. Razvili so ga leta 2011 pri podjetju Twitter [9] in je trenutno najpopularnejše tovrstno ogrodje. Glavni namen ogrodja je lajšanje oblikovanja spletnih dokumentov in deluje v celoti na odjemalčevi strani. Omogoča pa tudi preprosto izdelavo odzivnih aplikacij, kar pomeni, da se aplikacija samodejno prilagaja velikosti zaslona in je kot taka brez problemov uporabna tako na velikih računalniških zaslonih, kot tudi na manjših zaslonih mobilnih naprav. Bootstrap ima vgrajene funkcije za stolpce in vrstice (col in row) in zato omogoča precej lažjo postavitev elementov. Širina strani je

razdeljena na 12 delov, kot je prikazano na sliki 2.4, pri čemer lahko preprosto določamo širino naših elementov od 0 do 12.



Slika 2.4: Stolpci v ogrodju Bootstrap [10]

2.2.2 jQuery

jQuery je JavaScript knjižnica, ki poenostavi razvoj kode, ki se izvaja na odjemalčevi strani. V osnovi se uporablja za manipulacijo elementov DOM v dokumentu HTML [11]. Z njo izvajamo operacije (iskanje, izbiranje, spreminjanje elementov DOM) veliko preprostejše kot s samim JavaScriptom, poleg tega pa je koda veliko bolj berljiva, kot je vidno na sliki 2.5. DOM je predstavitev dokumenta oz. spletne strani, kjer so vsi elementi spletnega dokumenta zapisani hierarhično v drevesni strukturi. jQuery danes uporablja 65 odstotkov največjih spletnih strani na svetu in je podprta v vseh modernih brskalnikih [12].

```
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
```

Slika 2.5: Primer uporabe funkcij jQuery

2.2.3 .NET

.NET (dot NET) je ogrodje, ki ga je leta 2002 izdalo podjetje Microsoft in je bilo prvenstveno namenjeno uporabi na operacijskih sistemih Windows [13]. Med drugim nudi ogrodje podporo pri razvijanju spletnih storitev (ang. Web services). Vsebuje obsežno knjižnico razredov FCL in zagotavlja medopravilnost jezikov, kar pomeni, da lahko vsak programski jezik uporablja kodo napisano v drugih programskih jezikih, ki je skladna s specifikacijo CLI. Programi napisani za .NET se izvajajo v programskem okolju CLR, vrsti aplikacijskega virtualnega računalnika, ki je prav tako definiran v specifikaciji CLI. Izvajalsko okolje CLR pa med drugim zagotavlja zagotavlja varnost, upravljanje s pomnilnikom, ter upravljanje z izjemami.

Osnovna knjižnica FCL prinaša orodja za uporabniške vmesnike, podatkovni dostop, komunikacijo s podatkovnimi bazami, kriptografske postopke, funkcije za razvoj spletnih aplikacij, numerične postopke in spletno komunikacijo.

Za razvoj programov so na voljo številna integrirana razvojna okolja, od katerih sta najbolj razširjeni okolji Visual Studio in MonoDevelop.

Najnovejša različica, ki je uporabljena tudi v tem diplomskem delu, je .NET Core. To okolje je namenjeno razvoju aplikacij, ki so prenosljive med različnimi platformami, razvoju v oblaku, ter platformi UWP.

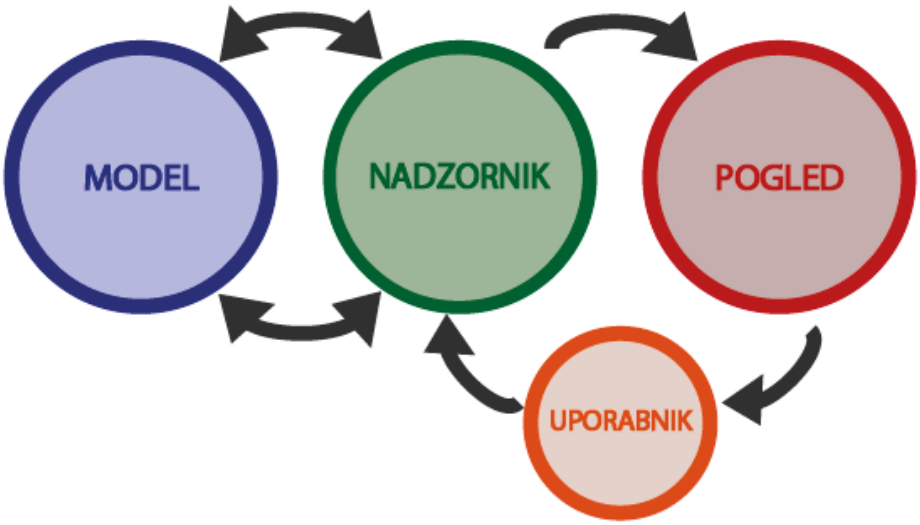
ASP.NET MVC

Model-pogled-nadzornik (ang. model-view-controller) je arhitekturni vzorec, ki narekuje strukturo uporabniških vmesnikov [14]. Loči aplikacijo na tri med seboj povezane dele in sicer (slika 2.6): model, pogled in nadzornik. Model predstavlja dejanske podatke in njihovo strukturo. Pogled je vizualni prikaz teh podatkov. Nadzornik pa interpretira uporabnikove ukaze in jih pretvori v ukaze za model ali pogled. ASP.NET MVC [15] je ogrodje, ki implementira MVC vzorec. V kontekstu našega dela je namenjen izdelavi in oblikovanju administracijske spletne strani in prikaza podatkov na njej. V naši nalogi smo za implementacijo pogledov uporabili Razor.

- **Model (ang. model)** je podatkovni vir, ki ga uporablja pogled. V tem primeru nastopa kot razred programskega jezika C#. Vsebuje vso podatkovno logiko, strukturo podatkov in njihove omejitve. V razvijalskem okolju Visual Studio se ta sestavljen model uporabi kot osnova na kateri zgradimo nadzornika s pomočjo ogrodja Scaffolding.

- **Pogled (ang. view)** nastopa v spletnih aplikacijah kot predloga HTML. Pogled sprejme model, ki ga, glede na to, kako ga razvijalec definira, potem tudi vizualno prikaže. Razen prikaza podatkov ne opravlja nobene druge funkcije.

- **Nadzornik (ang. controller)** nadzoruje tok dogodkov. Sam po sebi ne vsebuje nobene poslovne logike. Zadolžen je le za koordinacijo modela s pogledom. Nadzornik dobi zahtevek od uporabnika, ki ga nato izvrši, tako, da najde ustrezen model in ga preda pogledu, ki ga nato prikaže.



Slika 2.6: Delovanje arhitekture MVC

Web API

API oz. aplikacijski programski vmesnik je množica rutin, protokolov in orodij za izgradnjo programske opreme in aplikacij [16]. Omogoča oddaljen dostop do spletnih storitev in nam kot razvijalcem omogoča izgradnjo odjemalca za našo spletno storitev. .NET Web API je preprost način implementacije RESTful spletnih storitev [17], to je spletnih storitev, ki temeljijo na arhitekturnem slogu REST. V kontekstu naše naloge deluje podobno kot nadzornik MVC, a se ne ukvarja z dejanskim prikazom podatkov temveč le s predajo samih podatkov, ponavadi v obliki označevalnega teksta XML ali JSON.

2.2.4 Entity Framework

Ogrodje Entity je zbirka tehnologij, ki je namenjena razvoju podatkovno usmerjenih aplikacij [18]. Omogoča delo s podatkovnimi bazami na takšen način, da se razvijalec ne ukvarja z dejanskimi tabelami in se osredotoči na razvoj samih objektov v bazi. Cilj je odprava neusklajenosti med podatkov-

nimi modeli in programskimi jeziki, pri čemer pomaga tudi LINQ.

2.2.5 LINQ

LINQ (ang. language integrated query) je komponenta ogrodja .NET, ki v jezik C# doda zmožnost poizvedb SQL [19]. To nam omogoča, da lahko kar v kodi kličemo, sortiramo in na splošno upravljamo z objekti v naši podatkovni bazi. LINQ vsebuje vse operacije, ki jih podpira SQL, kot so: select, where, join, orderby, groupby itd.

2.3 Orodja

Na koncu bomo predstavili še orodja, v katerih je potekal razvoj sistema. Orodja sta pravzaprav le dva in sicer Visual Studio ter Android Studio. Slednji zadostuje našim potrebam pri razvoju preproste aplikacije za operacijski sistem Android, Visual Studio je pa zelo obširno orodje, ki nam omogoča upravljanje celotnega strežniškega dela sistema.

2.3.1 Visual Studio

Visual Studio je integrirano razvojno okolje (IDE), ki ga je razvilo podjetje Microsoft. Namenjeno je razvoju programov za operacijske sisteme Windows, spletnih strani, spletnih aplikacij, spletnih storitev in aplikacij na osnovi ogrodja .NET [20]

Vgrajen urejevalnik kode podpira v osnovi programske jezike C, C++, VB.NET, C# in F#. Naknadno se lahko z razširitvami doda podpora za jezike Python, Ruby, Node.js, ter mnoge druge. Z njim lahko urejamo tudi XML, HTML, JavaScript in CSS kodo.

V tem diplomskem delu je bila uporabljena različica Visual Studio 2015 Community Edition, ki dovoljuje uporabo za osebne in akademske namene.

2.3.2 Android Studio

Android Studio je integrirano razvojno okolje (IDE) namenjeno razvoju za platformo Android. Napovedano je bilo leta 2013, prva stabilna izdaja (verzija 1.0) pa je izšla decembra 2014. Okolje temelji na programski opremi IntelliJ IDEA podjetja JetBrains, a je v celoti preusmerjeno na razvoj za operacijski sistem Android [21]. Ustvarjeno je bilo z namenom zamenjave Eclipse Android Development Tools (ADT) kot primarnega razvojnega okolja za platformo Android. Vsebuje vsa orodja Android SDK Tools, katerih namen je oblikovanje, testiranje in razhroščevanje aplikacij. Android Studio omogoča tudi "oblikovalski pogled", ki v realnem času prikazuje izgled naše aplikacije. Po želji lahko tudi izbiramo med mnogimi različnimi emulatorji telefonov, na katerih lahko aplikacijo testiramo v različnih okoljih, z različnimi verzijami operacijskega sistema Android in z različnimi velikostmi zaslonov.

Gradle

Gradle je sistem za izgradnjo aplikacij. Temelji na konceptih Apache Ant in Apache Maven [22]. Njegov namen je odprava potrebe po ročni izgranji aplikacij in nam ta postopek občutno olajša. Vsebuje dinamičen pristop k upravljanju z odvisnostmi (ang. dependancy management). Omogoča tudi izgradnjo za razne spletne trgovine kot so trgovina Play [1], trgovina Amazon [2] itd. in omogoča gradnjo za razne namene (Debug, QA, Release).

[1] `https://play.google.com/store?hl=en`

[2] `https://www.amazon.com/gp/mas/get-appstore/android/ref=mas_rw_ldg`

Poglavje 3

Podatkovni model

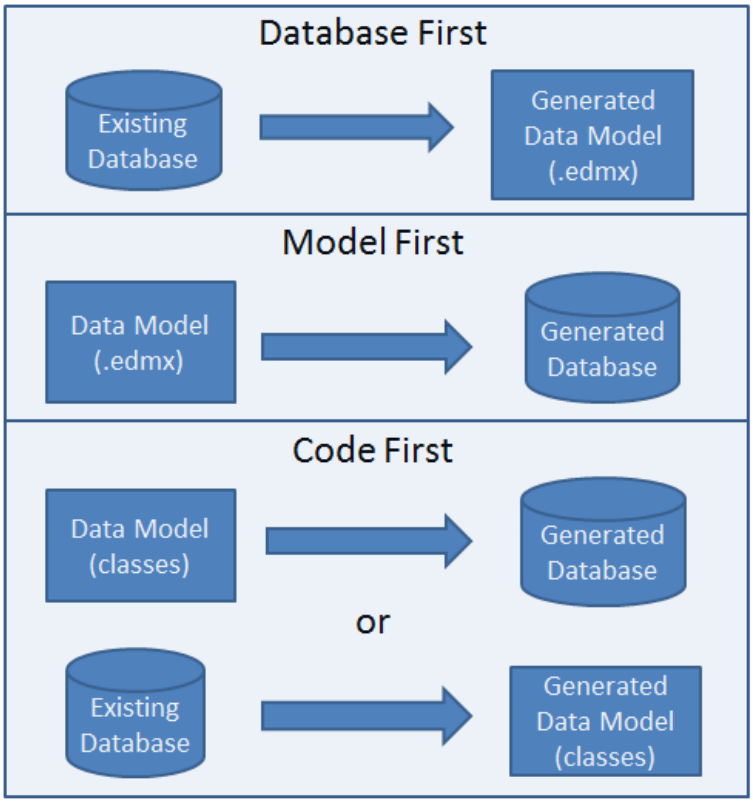
Pri izdelavi aplikacije s pomočjo ogrodja .NET Core ima razvijalec možnost povezati že ustvarjeno podatkovno bazo ali pa ustvari bazo kar med razvojem projekta. V našem projektu smo izbrali slednje. Pri tem smo si pomagali z ogrodjem Entity. Slika 3.1 prikazuje različne načine uporabe ogrodja Entity za delo s podatkovnimi bazami. Samo ogrodje Entity pa ne zadostuje vedno. V primeru, da želimo podatke sortirati, jih filtrirati ali pravzaprav izvesti kakršnokoli operacijo nad njimi, moramo uporabiti LINQ-to-SQL. .NET Linq že v osnovi vsebuje in nam omogoča izvajanje SQL poizvedb kar v kodi C#. Tega pristopa se poslužujemo tudi pri zbiranju statistike strank, saj nas zanima recimo kolikokrat se specifična stranka naroči pri določenem frizerju in katere storitve pri tem izbere. Tu uporabimo dinamično generirane LINQ poizvedbe.

3.1 Koraki razvoja podatkovnega modela

Pri razvoju podatkovnega modela smo uporabili ogrodje Entity, kjer smo model razvijali v dveh korakih.

3.1.1 Code-first

Pri pristopu "najprej-koda"(ang. Code-first approach) najprej ustvarimo razrede, ki služijo kot modeli objektov, ki jih želimo imeti v bazi. Ti razredi so zapisani v jeziku C# in jim kar v kodi določimo njihove atribute in omejitve. Ti modeli se nato s pomočjo migracij vstavijo v podatkovno bazo, kjer so samodejno urejeni v tabele SQL. Kasneje lahko podatkovno bazo urejamo neposredno ali tako, kot smo jo ustvarili, se pravi, da uredimo razrede z modeli in preko migracij posodobimo tabele.



Slika 3.1: Štiri načini uporabe ogrodja Entity [23]

3.1.2 ASP.NET Scaffolding

Ko so v bazi ustvarjene tabele je potrebno do teh podatkov tudi dostopati in izvajati nad njimi operacije CRUD (ustvari, preberi, posodobi, zbriši (ang. create, read, update, delete)). Visual Studio vsebuje ogrodje, ki omogoča samodejni način ustvarjanja nadzornikov in se imenuje Scaffolding [24]. Tako dobimo preproste nadzornike, ki skrbijo za prikaz, urejanje in dodajanje podatkov v bazo. V našem primeru, ker uporabljamo različico .NET Core, so to ASP.NET MVC nadzorniki, katerim kasneje dodamo tudi željene funkcije Web API. Ta pristop omogoča tudi izdelavo preproste spletne strani, na kateri je mogoče naše podatke prikazati, jih urejati in brisati iz podatkovne baze.

3.2 Opis modela

Vsaka tabela v naši podatkovni bazi je predstavljena v kodi z razredom, kot prikazuje slika 3.2. V teh razredih so zapisani vsi atributi naših entitet. Atributi pa imajo tudi v samem razredu določena validacijska pravila, ki določajo strukturo in omejitve vsakega polja. Entitete so sledeče:

- **Frizer** - v to tabelo mora administrator sistema ročno preko naše spletne administrativne konzole vnesti vse zaposlene frizerje v salonu.
  Atributi: ID, Koda, Ime, Priimek, StNarocil.

- **Stranka** - tu se shranjujejo vse stranke, ki preko našega sistema oddajo naročilo. Podatki, ki se shranijo v to tabelo so odvisni od tega, kaj stranka sama dovoli.
  Atributi: ID, Ime, Priimek, Telefon, Email, Geslo.

- **Storitev** - kot pri entiteti Frizer, mora administrator storitve preko spletne aplikacije vnesti v podatkovno bazo ročno.
  Atributi: ID, ImeStoritve, CenaStoritve, ČasStoritve, Frizer1, Frizer2, Frizer3.

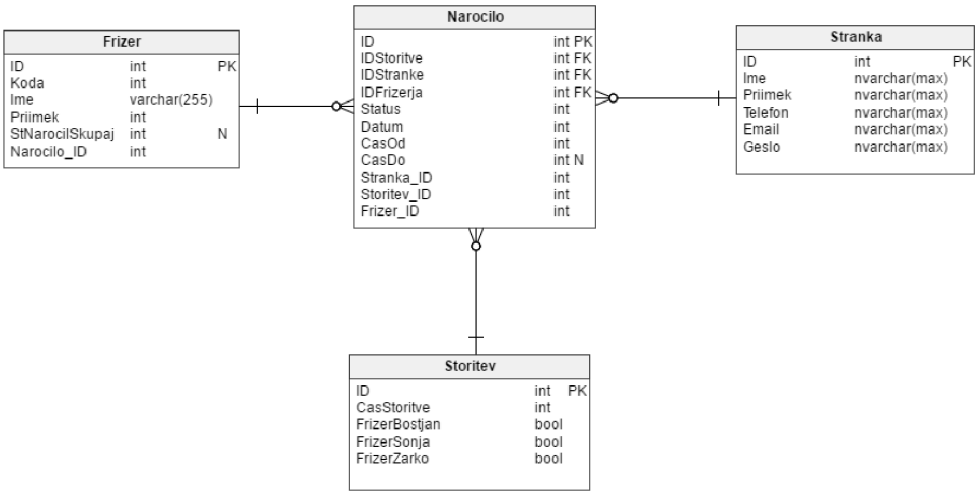
- **Naročilo** - vsakič, ko stranka odda naročilo, se le ta shrani v nov objekt Narocilo, ki se nato samodejno vnese v podatkovno bazo, poleg tega pa se pošlje izbranemu frizerju obvestilo na njegov uporabniški račun v aplikaciji.
  Atributi: ID, IDStoritve, IDFrizerja, Status, Datum, CasOd, CasDo.

```
public class Frizer
{
    public int ID { get; set; }

    [Required]
    public int Koda { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z''-'\s]*$"), Required, StringLength(30)]
    public string Ime { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z''-'\s]*$"), Required, StringLength(30)]
    public string Priimek { get; set; }

    [Display(Name = "Vsa Naročila")]
    public int StNarocilSkupaj { get; set; }
}
```

Slika 3.2: Primer razreda, ki ga vstavimo v podatkovno bazo kot tabelo

3.3 Shema podatkovne baze

Naša shema, ki jo vidimo na sliki 3.3, je razmeroma preprosta. Osrednja točka celotne baze je tabela "Narocilo". Ta služi kot vezava med ostalimi tabelami saj vsebuje identifikacijo (ID) frizerja, storitve in stranke.



Slika 3.3: Pregled tabel v podatkovni bazi in relacije med njimi

Poglavje 4

Razvoj spletne aplikacije

Celoten rezervacijski sistem je sestavljen iz dveh ločenih aplikacij, ki med seboj le komunicirata s pomočjo arhitekturnega sloga REST. Spletna aplikacija je napisana v okolju Visual Studio in temelji na ogrodju .NET, ki v svoji najnovejši različici .Net Core združuje pristop MVC in ogrodje Web API. To nam je omogočilo sestaviti aplikacijo, ki istočasno skrbi za komunikacijo z Android aplikacijo ter prenaša podatke iz podatkovne baze na spletno stran. Slednja služi kot preprosta spletna administracijska konzola in zbirka statistike o strankah, ki se naročajo preko tega sistema. Na drugi strani pa imamo Android aplikacijo, ki je namenjena izključno končnim uporabnikom in deluje kot nekakšen portal za frizerski salon.

V tem poglavju bomo opisali razvoj spletne aplikacije. Z izjemo oddajanja in potrjevanja naročil, skrbi spletna aplikacija za izvajanje celotne poslovne logike sistema. Zadolžena pa je tudi za spletni prikaz vseh podatkov, ki se pretakajo med zaposlenimi v frizerskem salonu in strankami. Tu lahko administrator sistema po potrebi dodaja, spreminja in briše frizerje in storitve, ki jih frizerski salon ponuja. Spletna stran služi tudi kot vizualni prikaz statistike. Tako lahko lastnik salona takoj najde stranke z največ obiski in najpopularnejše storitve v svojem salonu.

Ob prihodu .NET Core je podjetje Microsoft predstavilo nekoliko spremenjen način ustvarjanja spletnih aplikacij. V .NET 4.6 in prejšnjih različicah

je bila uporaba Web API in MVC strogo ločena, saj je vsaka storitev uporabljala za prikaz podatkov lastnega nadzornika. ASP.NET MVC je namenjen izdelavi spletnih strani, zato MVC nadzornik vrača poglede (View). Web API na drugi strani pa je namenjen komunikaciji z drugimi aplikacijami oz. odjemalci. V tem projektu sta pomembna torej oba pristopa, saj želimo imeti administracijo na preprosti spletni strani, ob istem času pa mora aplikacija omogočati dostop do podatkovne baze tudi naši Android aplikaciji. V .NET Core sta MVC in Web API združena v okviru enega nadzornika in je zato veliko lažje voditi en projekt, ki omogoča uporabo obeh pristopov. Primera dveh funkcij, ki vračata iste podatke, pri čemer ena vrača pogled, druga pa objekt JSON, vidimo na slikah 4.1 in 4.2

Spletna aplikacija združuje MVC in Web API. To pomeni, da je projekt strukturiran tako, da vključuje tako kodo, ki se izvaja na strani strežnika, kot tudi kodo, ki se izvaja na strani odjemalca. Prvotno je bilo v načrtu to dvoje ločiti, a ker je del naše spletne aplikacije, ki se izvaja na odjemalčevi strani, relativno majhen, je bila izbira tovrstne strukture precej smiselna. To sicer pomeni nekoliko slabšo fleksibilnost in manjšo modularnost sistema, a istočasno olajša komunikacijo med posameznimi deli. Ker pa je naš odjemalski del zgolj administracija, ni bilo potrebno veliko oblikovanja same spletne strani, zaradi česar smo se bolj osredotočili na samo preprostost in uporabnost. Takšen način dela spodbuja tudi Microsoft, saj je struktura projekta v predlogi, ki jo ponuja Visual Studio za razvoj projektov na osnovi ogrodja .NET Core, ravno takšna. Vsa poslovna logika poteka v samih modelih objektov, za prikaz pogledov pa skrbi nadzornik, ki vsebuje tudi vso kodo za prenos podatkov preko Web API.

```
public async Task<IActionResult> Index()
{
    return View(await _context.Storitev.ToListAsync());
}
```

Slika 4.1: Prikaz klica ukaza GET, ki vrača pogled

```
public async Task<IActionResult> GetAndroid()
{
    var frizerji = from f in _context.Frizer
                   select f;

    return Json(await frizerji.ToListAsync());
}
```

Slika 4.2: Prikaz klica ukaza GET, ki vrača JSON

4.1 Uporabniški vmesnik

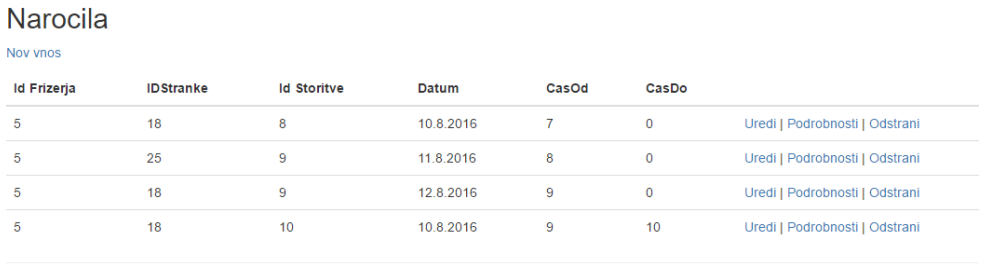
Uporabniški vmesnik naše administracijske spletne strani je zelo preprost. Sestavljen je s pomočjo ogrodja Bootstrap, saj to precej olajša oblikovanje odzivne spletne strani, ki deluje na vseh velikostih zaslonov. Veliko pomaga tudi pri sami postavitvi elementov in dodajanju izbirnega menija v vmesnik. Tako imamo na vsaki podstrani v aplikaciji na vrhu meni, ki prikazuje vse možne podstrani, v zgornjem levem kotu pa logotip frizerskega salona. Poleg logotipa nismo tu dodajali nobenih drugih podatkov o podjetju ali podobnih informacij, ki so neuporabne za lastnika frizerskega salona, saj je stran namenjena le njemu. Zaradi tega tudi ni pozdravne strani in se namesto tega takoj ob zagonu pojavi pogled vseh zaposlenih.

4.2 ASP.NET MVC

Kot smo omenili, je MVC nadzornik zasnovan tako, da njegove funkcije vračajo poglede. Zato se pri gradnji nadzornika s pomočjo ogrodja Scaffolding iz modela ustvari pet spletnih strani Razor s končnico .cshtml, ki implementirajo ukaze REST (GET, POST, PUT, DELETE). Te strani so:

- **/BuusAdmin/Narocila/**, kjer so vsi podatki izbranega modela oz. izbrane tabele iz podatkovne baze prikazani v tabeli. Dodali smo še sortiranje podatkov. Uporablja metodo GET, da pridobi podatke iz

naše podatkovne baze. Na sliki 4.3 lahko vidimo primer prikaza vseh naročil.

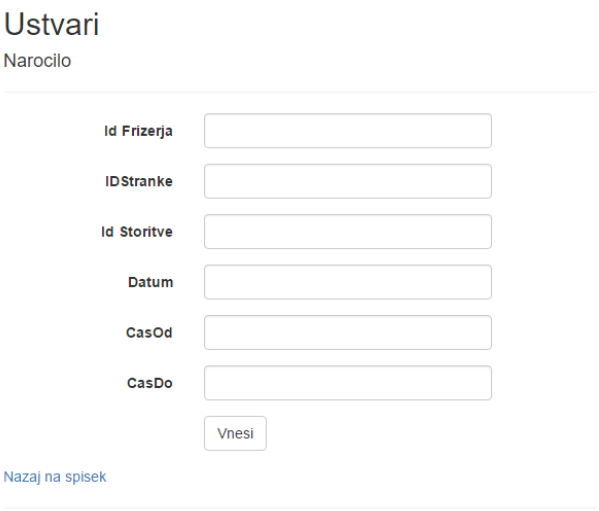


Slika 4.3: Pogled Index.csthml

- **/BuusAdmin/Narocila/Details/*id***, ki prikaže posamezen vnos na bolj pregleden način in prav tako uporablja metodo GET za pridobitev podatkov le enega vnosa. Na sliki 4.4 lahko vidimo podroben prikaz posameznega naročila.

Slika 4.4: Pogled Details.cshtml

- **/BuusAdmin/Narocila/Create**, ki prikaže obrazec za vnos novega elementa v tabelo in ga preko metode POST tudi doda v podatkovno bazo. Slika 4.5 prikazuje obrazec za vnos elementa v tabelo.

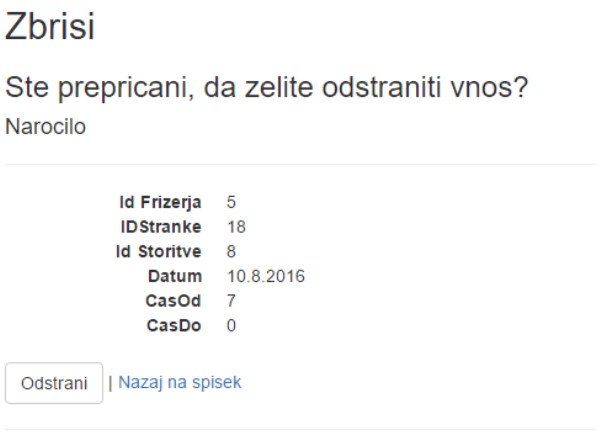


Slika 4.5: Pogled Create.cshtml

- **/BuusAdmin/Narocila/Edit/*id***, ki najprej z metodo GET vse podatke enega vnosa pridobi in nato ob uporabnikovi potrditvi uporabi metodo PUT, da vnos v bazi spremeni na željene vrednosti. Slika 4.6 prikazuje obrazev, ki omogoča urejanje obstoječega vnosa.

Slika 4.6: Pogled Edit.cshtml

- **/BuusAdmin/Narocila/Delete/*id***, ki uporablja metodo DELETE z namenom, da željen vnos iz podatkovne baze odstrani.



Slika 4.7: Pogled Delete.cshtml

4.3 Web API

Glavni namen spletne aplikacije je vzpostaviti API, ki je odgovoren za komunikacijo med strežnikom Windows in odjemalcem Android. Če bi bil naš cilj razviti le spletno aplikacijo, na katero bi uporabniki lahko vstopali, bi bil ta del nepotreben. Ker pa je eden izmed glavnih ciljev naše naloge postaviti izvirno aplikacijo Android, ki služi kot odjemalec in uporabniški vmesnik našega sistema za stranke, nam le nadzornik MVC ne bo zadostoval. Zato smo dodali svoje krmilniške funkcije, ki namesto pogledov vračajo in sprejemajo preproste objekte JSON. Te objekte lahko nato v Android delu sistema z lahkoto preberemo in jih uporabnikom tudi izpisujemo.
Primer objekta JSON enega naročila:
(”ID”:17, ČasDo”:0, ČasOd”:7, ”Datum”:”10.8.2016”, ”IDFrizerja”:5, ”IDStoritve”:8, ”IDStranke”:18, Štatus”:3).

Podobno kot nadzornik MVC, so tudi Web API krmilniške funkcije preproste operacije CRUD, pri čemer v naši aplikaciji uporabljamo le metodi

GET, ki vrne preprost objekt JSON in POST, ki sprejme preprost objekt JSON, ki pa je trenutno v obliki golega besedila in bi ga bilo potrebno v končni verziji, ki bi bila uporabljena v realnem okolju, kriptirati.

4.4 Varnost

Varnost v spletni aplikaciji je precej osnovna. Za dostop do same spletne administracije je potrebno uporabniško ime in geslo, ki je določeno vnaprej. V tem delu ne vključujemo možnosti registracije ali kreiranja novih uporabnikov, saj je aplikacija zasnovana tako, da ima dostop do vseh podatkov le administrator, kar je v tem primeru ponavadi lastnik frizerskega salona. S tem se na enostaven način izognemo kar nekaj možnim vrstam napadov na naš sistem.

Poglavje 5

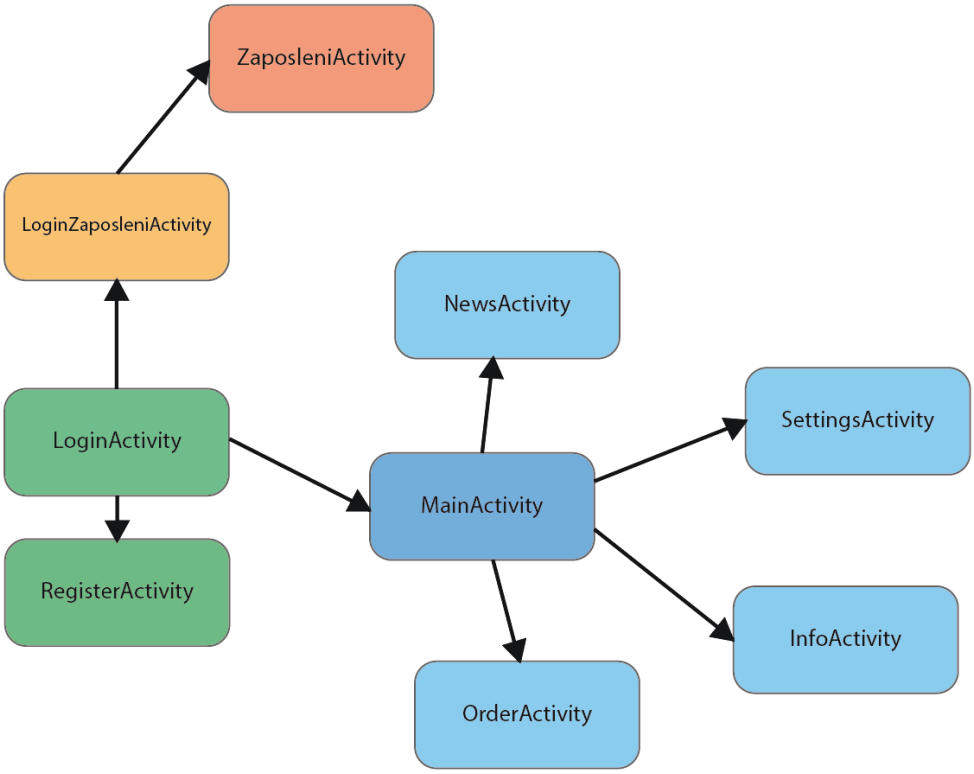
Razvoj Android aplikacije

Drugi del našega sistema je mobilna aplikacija, ki se izvaja operacijskem sistemu Android. Aplikacija je ključni del celotnega sistema, saj ponuja rezervacijo terminov iz mobilnega telefona. Čeprav bi bilo mogoče narediti spletno aplikacijo, s katero bi stranke upravljale preko spletnega brskalnika, smo se odločili, da ustvarimo kar izvirno (ang. native) Android aplikacijo, saj je uporabniška izkušnja v primerjavi s prvotnim predlogom precej boljša. To nam omogoča hitrejše odzivne čase, večjo intuitivnost uporabe in bolj modularen pristop k izdelavi celotnega sistema. Za glavne cilje mobilne aplikacije smo si zadali:

- intuitivnost,
- preglednost,
- odzivnost,
- multifunkcionalnost,
- preprostost nadgradnje,
- oblikovanje v skladu z najnovejšimi smernicami (Material design).

Razvita aplikacija Android je pravzaprav sestavljena iz dveh ločenih delov, ki opravljata različne naloge. V primeru, da se prijavi nov uporabnik oz.

uporabnik, za katerega že predhodno vemo, da je stranka frizerskega salona, se odpre aplikacija, ki služi kot predstavitvena aplikacija salona. Vključuje podatke o salonu in frizerjih ter omogoča uporabniku, da si ogleda najnovejše novice in dogodke. Glavni del tega sklopa aplikacije pa je sistem, ki omogoča izbiro in rezervacijo termina za poljubne frizerske storitve pri željenem frizerju. Drugi del aplikacije je namenjen le določenim uporabnikom oz. v našem primeru frizerjem frizerskega salona. Tu dobi frizer obvestilo o novem naročilu. Naročilo lahko tudi zavrne in predlaga nov termin ali vzpostavi neposredno komunikacijo med njim in stranko preko telefonskega klica ali sporočila SMS. V primeru, da mu termin ustreza, ga preprosto potrdi in stranka dobi o tem obvestilo. Možna prehajanja med aktivnostmi v aplikaciji lahko vidimo na sliki 5.1



Slika 5.1: Struktura vprašalnika za rezervacijo

Vse povezave z našim strežnikom smo opravili s knjižnico OkHttp, s katero je enostavno sestaviti odjemalca HTTP in se povezati na določen spletni naslov URL. Ker imamo postavljeno na strežniku spletno aplikacijo pa lahko preko samega naslova URL dobimo podatke, ki jih želimo. Za povezovanje s strežnikom smo napisali samostojni razred, ki razširja AsyncTask. To je pomembno, saj tovrstnega povezovanja ni mogoče opravljati na glavni niti in je potrebno to izvajati v ozadju. Aplikacija zahteva od uporabnika sprejetje nekaterih dovoljenj in sicer uporabo interneta, saj se aplikacija povezuje na oddaljen strežnik, in uporabo osebnih podatkov, saj le te pošilja na strežnik in so ključni tako za komunikacijo med strankami ter zaposlenimi, kot tudi za preprečevanje nezaželjenega izkoriščanja same aplikacije.

5.1 Aplikacija namenjena strankam

Del aplikacije namenjen strankam je oblikovan kot predstavitvena aplikacija za frizerski salon. Sestavljen je iz strani za novice, ki jih lahko dodaja in spreminja le administrator, strani z informacijami o salonu, ki vsebuje podatke o lokaciji, podatke o zaposlenih in njihove kontaktne informacije, ter splošne podatke o frizerskem salonu. Glavni del tega sklopa aplikacije pa je seveda sistem za rezervacijo termina, pri čemer smo uporabili predelano knjižnico Wizard Pager, ki deluje kot čarovnik in uporabnika popelje skozi celoten postopek rezervacije. Stranka tu izbere svojega frizerja, željene storitve in pa na koncu poljuben termin. Slednji del je najpomembnejši in najkompleksnejši. Koledar, ki ga vidi stranka, je sinhroniziran s frizerjevim javnim Google koledarjem, tako, da lahko stranka vidi, kdaj so prosti termini. Storitve, ki jih je stranka prej izbrala, pa narekujejo koliko časa bo obisk trajal, kar potem tudi samodejno omeji možnost izbire termina. Ta del aplikacije je sestavljen iz le štirih aktivnosti: OrderActivity (aktivnost za naročila), NewsActivity (aktivnost za novice), SettingsActivity (aktivnost za nastavitve) in InfoActivity (aktivnost za informacije). Te aktivnosti se potem glede na potrebe delijo na fragmente (komponenta Fragment) ali pa ob določenih ukazih uporabnika

prikazujejo pogovorna okna (komponenta Dialog).

5.1.1 Aktivnost OrderActivity

Ta aktivnost je ključni del celotne aplikacije. Za izvedbo vprašalnika oz. čarovnika smo uporabili knjižnico Wizard Pager. Deluje tako, da v ločenem Java razredu (WizardModel) določimo strukturo celotnega vprašalnika, ki nato samodejno poskrbi za strani v čarovniku. Za naše potrebe pri izbiri datuma in časa termina je bilo potrebno po meri posebej sestaviti svojo stran DatePage, tako, da od uporabnika kot rezultat sprejme rezultat našega koledarja. Osnovno strukturo vprašalnika potem določimo na preprost način, kot prikazuje slika 5.2. Sam koledar je oblikovan tako, da istočasno prikazuje dni v tednu in čas v dnevu, za kar smo uporabili knjižnico WeekView. Sprva je bila načrtovana le preprosta izbira datuma s komponento DatePicker in časa s komponento TimePicker, a se je uporaba knjižnice WeekView izkazala kot najbolj intuitivna in pregledna izbira.

```
return new PageList(
        new SingleFixedChoicePage(this, "Izberi Frizerja").setChoices(
                frizersStringArray)
                .setRequired(false),
        new SingleFixedChoicePage(this, "Izberi Storitve").setChoices(
                storitveStringArray)
                .setRequired(true),
        new DatePage(this, "Izberi Datum")
                .setRequired(true)
);
```

Slika 5.2: Struktura vprašalnika za rezervacijo

5.1.2 Aktivnost NewsActivity

Aktivnost deluje kot preprost spisek novic. Sestavlja ga pogled ListView, ki ga napolnimo z našim adapterjem NoviceAdapter, ki vsebuje po meri sestavljene elemente.

5.1.3 Aktivnost InfoActivity

V tej aktivnosti so prikazane informacije. S pomočjo postavitve TabLayout, smo razdelili aktivnost na tri dele oziroma fragmente, pri čemer prvi vsebuje informacije o samem frizerskem salonu, drugi informacije o zaposlenih in tretji kontaktne informacije ter lokacijo salona.

5.1.4 Aktivnost SettingsActivity

Tu lahko uporabnik spreminja svoje uporabniške podatke. V primeru, da uporabnik shrani podatke, se kliče ustrezna funkcija preko klica metode PUT, tako da se v podatkovni bazi na strežniku podatki zamenjajo.

5.2 Aplikacija namenjena zaposlenim

Ko stranka odda svoje naročilo, dobi izbran frizer na svoji mobilni napravi obvestilo, ki vsebuje potrebne podatke o stranki, izbranih storitvah in izbranem terminu. Zaposleni lahko nato naročilo preprosto sprejme, če pa se z njim ne strinja, ga ima možnost zavrniti. V primeru zavrnitve se mu ponudi možnost vzpostavitve neposredne komunikacije s stranko v obliki telefonskega klica, sporočila SMS ali sporočila preko elektronske pošte. V kolikor naročilo sprejme, dobi stranka nazaj obvestilo in se na frizerjevem koledarju dogodek obarva rdečo (Slika 5.3), na koledarju, kjer stranke oddajajo naročila, pa se izbrani termin označi kot zaseden. V prihodnje bi lahko za boljšo izkušnjo dogodke sinhronizirali s frizerjevim Google koledarjem in posledično tudi koledarjem Business Calendar, ki ga zaposleni v salonu uporabljajo za načrtovanje terminov službenih dogodkov.

Slika 5.3: Prvi pogled predela aplikacije za zaposlene

5.3 Zasnova uporabniškega vmesnika

Uporabniški vmesnik je zasnovan tako, da je kolikor se da preprost in intuitiven. Vmesnik je precej osnoven, a je zaradi tega precej preprost za uporabo in uporabnik ne išče željenih funkcij. Sam izgled vmesnika pa temelji na Googlovih najnovejših smernicah oblikovanja imenovanih Material design. Barve so izbrane po predlagani paleti (Slika 5.4), pri čemer se skozi celotno aplikacijo uporabljajo le štiri različne barve. Končni cilj je vezati barvno paleto na trenutno izbrano temo na uporabnikovem telefonu oz. prepustiti izbiro barv samim uporabnikom v meniju Nastavitve. Sam izgled aplikacije je trenutno precej osnoven in bi ga bilo potrebno še bolj dodelati za komercialno uporabo.

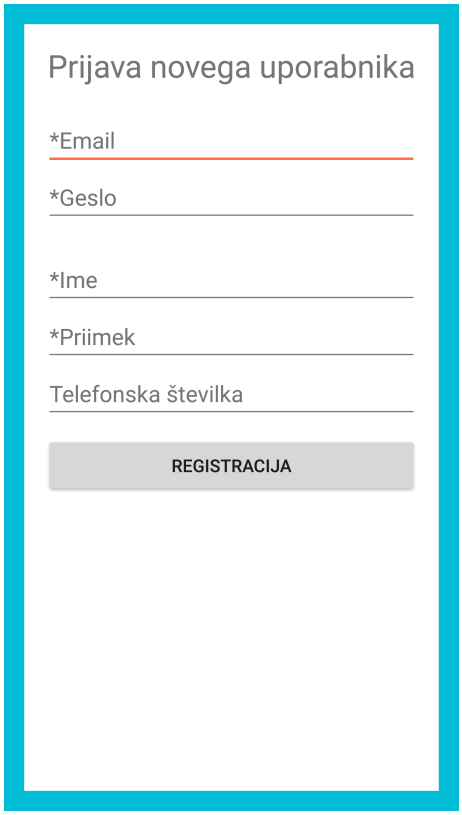
Slika 5.4: Izbrana barvna paleta

5.4 Varnost

Varnost v Android aplikaciji je nekoliko večja skrb, kot pri spletni aplikaciji. Ta del sistema je namenjen končnim uporabnikom in je zato veliko bolj odprt za izkoriščanje. Uporabniki morajo pri prvem zagonu aplikacije ustvariti nov uporabniški račun s preprostim obrazcem, kot prikazuje slika 5.5. Pred dokončno izdajo aplikacije bi bilo sicer smiselno dodati še sistem za avtentikacijo s pošiljanjem potrditvene elektronske pošte ali prijavo z računom Google. To bi vsaj nekoliko omejilo možnost kreiranja večje količine ponarejenih računov in rezervacij obiskov.

Komunikacija med odjemalcem Android in strežnikom Windows pa je trenutno z varnostnega stališča najbolj problematična, saj poteka v golem teksu in bi ga bilo potrebno v dokončni aplikaciji kriptirati.

Pri prijavi novega uporabnika stranke same izberejo, katere svoje podatke želijo poslati pri naročilu, a je večina podatkov nujnih, saj jih potrebuje frizer pri potrjevanju oziroma zavračanju naročila.



Slika 5.5: Pogled za registracijo novega uporabnika

Poglavje 6

Predstavitev uporabe

V tem poglavju bomo podrobneje predstavili delovanje celotnega sistema z vidika končnih uporabnikov. Ker je dandanes uporaba pametnih telefonov že tako splošno razširjena, smo se odločili narediti uporabniški del sistema zgolj na mobilnih napravah. Od vseh uporabnikov mobilnih telefonov jih večina uporablja operacijski sistem Android, zato smo se odločili, da razvijemo uporabniški sistem oziroma mobilno aplikacijo na tem sistemu.
Ob prvem zagonu aplikacije se pojavi pogled za prijavo. V primeru, da uporabnik že ima uporabniški račun v našem sistemu, le vnese uporabniško ime in geslo, s čimer je prijava končana. Ostali morajo svoj račun najprej ustvariti, kar naredijo tako, da pritisnejo na gumb "Registracija", kjer se jim prikaže obrazec, v katerega vnesejo svoje podatke, ki se potem na strežniku samodejno shranijo v podatkovno bazo. Postopek za zaposlene v salonu pa je nekoliko drugačen. Najprej pritisnejo na gumb "Prijava za zaposlene"in nato v sledeči aktivnosti vnesejo v polje svojo šifro, ki jim je bila dodeljena s strani administratorja. To popelje uporabnika v popolnoma drug del aplikacije, kjer imajo frizerji vpogled v svoja naročila.

6.1 Stranke

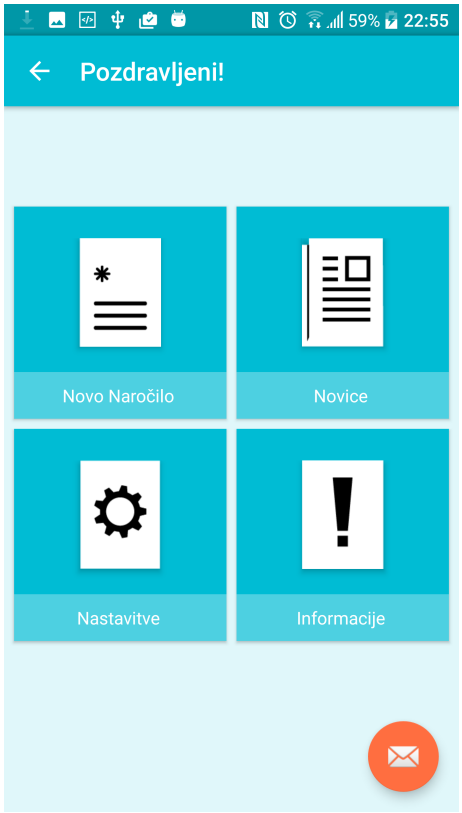
Med stranke uvrščamo vse, ki se v mobilno aplikacijo prijavijo in niso na vnaprej določenem spisku zaposlenih v frizerskem salonu. Ti dve skupini ločimo tako, da imamo dve ločeni aktivnosti za prijavo uporabnikov - eno za stranke in eno za zaposlene. Tako v aplikaciji ne moremo zamešati strank in frizerjev, saj mora stranka svoj račun ustvariti, frizer pa od lastnika salona oz. administratorja aplikacije dobi vnaprej določeno kodo in je, tudi kar se podatkovne baze tiče, v aplikacijo prijavljen kot "Frizer".

6.1.1 Glavni meni

Po vpisu se uporabniku odpre primarni pogled aplikacije (slika 6.1), ki obsega štiri podstrani:

1. Novo naročilo
2. Novice
3. Informacije
4. Nastavitve

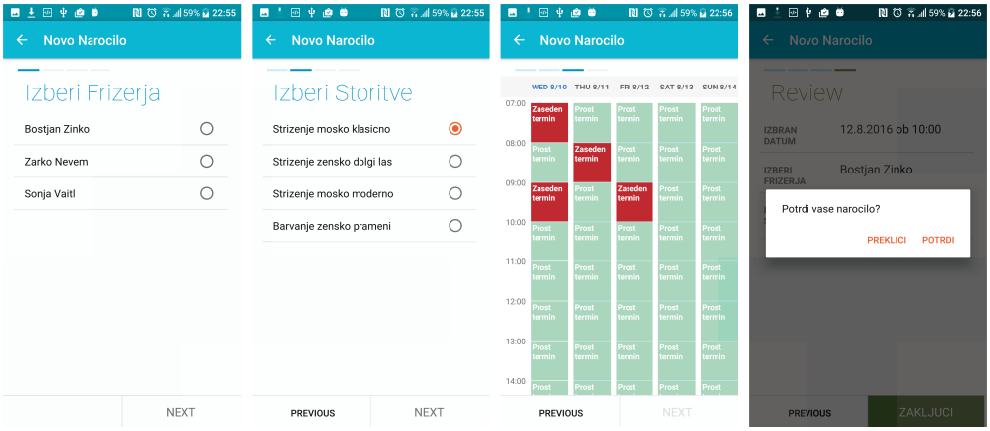
Glavni namen dela aplikacije za stranke je rezervacija termina pri izbranem frizerju. Kljub temu pa je širše gledano aplikacija mišljena kot centralni portal za frizerski salon, saj lahko aplikacija služi kot prvi stik novih potencialnih strank s salonom, tako da je dobro imeti še druge predele, kjer lahko stranka pridobi nekaj informacij o salonu in zaposlenih v njem, lokacijo in kontaktne informacije. Poleg tega je dobro stranke obveščati o novostih v salonu, kot so razni popusti, novi izdelki, novi frizerji, itd., kar dosežemo z aktivnostjo "Novice", kamor lahko administrator sam dodaja željeni material. Tako aplikacija na nek način že skoraj deluje kot preprost CRM, sploh če upoštevamo, da imamo možnost spremljanja statistik vseh strank v svoji bazi.



Slika 6.1: Glavni meni aplikacije

6.1.2 Novo naročilo

Izbira novega naročila odpre čarovnika, ki pelje uporabnika skozi celoten proces rezervacije termina. Sestavljen je tako, da vsaka izbira v posameznem delu čarovnika vpliva na ponujene izbire v naslednjih korakih. Torej, ko uporabnik izbere željenega frizerja, se v naslednjem koraku samodejno filtrirajo storitve, ki jih ta frizer lahko opravlja. Logika te komponente se izvaja na strežniku. Po izbiri storitve oz. večih storitev se še dodatno določi kako dolgo naj bi obisk trajal. Obe odločitvi pa vplivata na zadnji korak procesa, v katerem uporabnik izbira dejanski željeni datum in uro obiska. Ta izbira je prikazana v tedenskem pogledu, saj je najbolj pregleden in priročen. V pogledu se samodejno odstranijo tisti termini, kjer frizer ni v službi, tisti, kjer ima že zapolnjen čas, in tisti, v katerih je premalo časa za izvedbo izbranih storitev. Po izbiri termina se stranki prikaže še pregled vseh izbir in možnost potrditve naročila (slika 6.2).



Slika 6.2: Pogled za rezervacijo termina

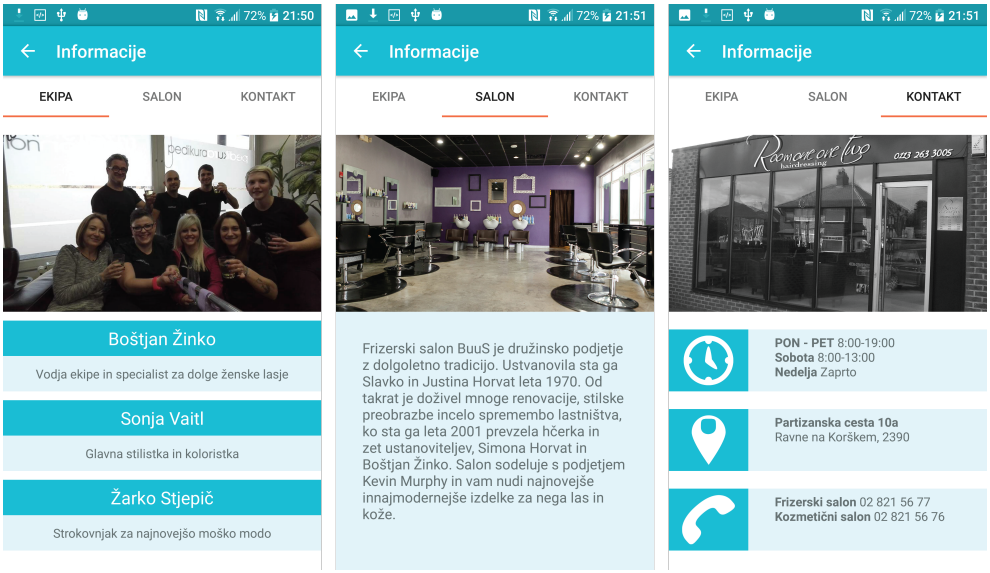
6.1.3 Novice

Pogled novic je pomemben del same aplikacije (slika 6.3). V tem delu lahko lastnik frizerskega salona oglašuje najnovejše dogajanje v salonu, kot na primer znižanja, nove izdelke, nove storitve itd. Glavni namen tega dela je, da služi kot način privabljanja novih uporabnikov k aplikaciji in pripomore k vračanju obstoječih uporabnikov. Administrator članke dodaja, spreminja in briše preko administracijske spletne aplikacije.

Slika 6.3: Pogled za novice

6.1.4 Informacije

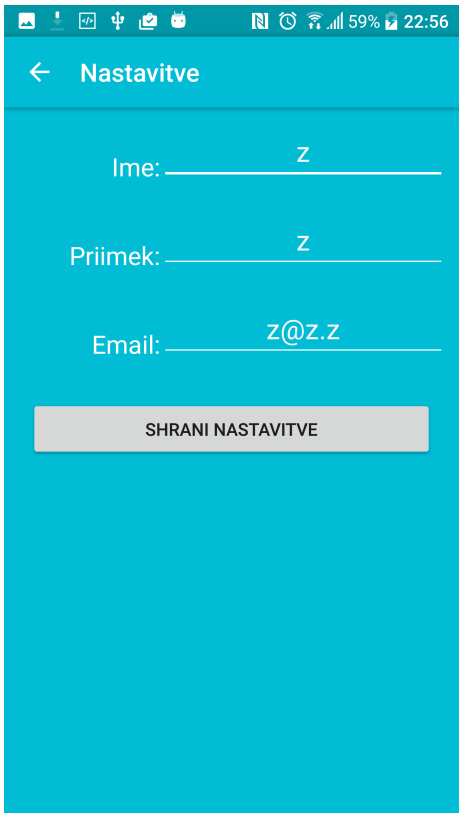
Ta pogled (slika 6.4) je namenjen predvsem novim strankam salona. Aktivnost je ločena na tri predele z zavihki. V zavihku Ekipa (slika 6.4, levo) lahko uporabnik dobi nekaj podatkov o zaposlenih v frizerskem salonu. V zavihku Salon (slika 6.4, sredina) so zapisane informacije in kratka zgodovina o samem salonu, v zavihku Kontakt (slika 6.4, desno) pa so kontaktne informacije in lokacija podjetja.



Slika 6.4: Pogled za informacije o salonu

6.1.5 Nastavitve

Uporabnik lahko v okviru pogleda z nastavitvami (slika 6.5) spreminja nastavitve svojega računa in jih shrani neposredno v podatkovno bazo na strežniku.



Slika 6.5: Pogled za nastavitve uporabniškega računa

6.2 Frizer

Zaposleni frizerji frizerskega salona so v sistem vnešeni vnaprej in se jim pri prijavi v aplikacijo odpre povsem drug pogled kot ostalim uporabnikom. Je veliko manj obsežna aplikacija, kot tista namenjena ostalim uporabnikom, saj je njen namen veliko ožji. Prva aktivnost, na katero uporabnik naleti, je kar koledar, ki je podoben tistemu pri strankini rezervaciji termina, in je prikazan s pomočjo knjižnice WeekView.

V tem pogledu se frizerju izrišejo vsa njegova naročila, tako tista, ki jih je že sprejel, kot tista, ki čakajo na njegovo odobritev. Ob pritisku na slednje se, kot lahko vidimo na sliki 6.6, odpre novo pojavno okno, ki mu da na izbiro potrditev izbrane rezervacije ali zavrnitev. Pri zavrnitvi mora frizer izbrati ali stranko preko telefona pokliče, ji pošlje sporočilo SMS ali elektronsko pošto, z namenom, da se dogovorita za nov termin. Tako dobi frizer, ki aplikacijo zažene, takoj vpogled na svoje rezervacije. V spodnjem desnem kotu je plavajoči gumb (Floating Action Button), ki prikazuje število novih obvestil. Pritisk na ta gumb mu prikaže vsa nova naročila, ki so jih stranke oddale in jih še ni potrdil ali zavrnil (pri zavrnitvi se mora s stranko zmeniti za nov termin).

Celoten potek uporabe v tej aplikacije teče v eni sami aktivnosti, po potrebi pa se uporablja tudi po meri sestavljena pogovorna okna (komponenta Dialog), ki omogočajo multifunkcionalnost te aktivnosti. Cilj je frizerja čim manj zamotiti in mu omogočiti enostavno urejanje svojih naročil v čim krajšem možnem času. V prihodnje je v tem pogledu načrtovana tudi popolna sinhronizacija s koledarjem Google Calendar, kar bo omogočalo boljšo integracijo naše aplikacije z že obstoječimi koledarji, ki se jih zaposleni trenutno poslužujejo.

Slika 6.6: Dialog za sprejem oziroma zavrnitev naročila

Poglavje 7

Sklepne ugotovitve in nadaljnje delo

Cilj diplomskega dela je bil razviti sistem, ki pomaga pri naročanju strank v frizerskem salonu. Opisali smo metode dela, orodja in tehnologije, predstavili delovanje sistema in podali primer njegove uporabe z vidika zaposlenih ter strank frizerskega salona. Poudarek smo dali na preprostost uporabe in visoko preglednost. Sistem je namenjen strankam, ki rade umirjeno in premišljeno oddajo svojo rezervacijo, kar včasih pri neposredni govorni komunikaciji ni mogoče. To smo dosegli tako, da smo razvili aplikacijo za operacijski sistem Android. Ker pa je nujno, da lahko aplikacijo administrator sistema preprosto spremlja in podatke v njej spreminja, je bilo potrebno razviti tudi spletni API za manipulacijo s podatki v podatkovni bazi, ter administracijsko konzolo, ki je dovolj preprosta, da jo lahko uporablja nekdo, ki nima ozadja v računalništvu. Prvi del sistema je spletna aplikacija, ki jo lahko odpremo v brskalniku. Ker ni potrebe po več uporabnikih, je dostopna le z enim uporabniškim imenom in geslom, saj na ta način ne žrtvujemo varnosti. Z njo lahko lastnik frizerskega salona ureja novice, dodaja nove zaposlene in storitve, ter spremlja statistiko obiskov vseh svojih strank. Za ta del smo uporabili orodje Visual Studio, ki uporablja jezik C# in ogrodje .NET.

Drugi del sistema je aplikacija Android, ki je večnamenska. Tako na eni strani omogoča zaposlenim v salonu hitro in učinkovito naročanje svojih strank in pregledno spremljanje svojega urnika. Na drugi strani pa je aplikacija centralni portal za stranke frizerskega salona. Tu lahko dobijo potrebne informacije o salonu, spremljajo dogajanje v njem in, najpomembnejše, oddajajo rezervacije terminov pri svojem frizerju. Glavna prednost tovrstnega pristopa k rezervaciji terminov je možnost prikaza razpoložljivega časa. V nasprotju s telefonskim klicem za rezervacijo ima stranka vpogled v vse možne termine in si lahko bolje načrtuje svoj obisk. Prav tako je zaradi dodanih funkcij v Android aplikaciji to dobra predstavitev za frizerski salon, ki se lahko dopolnjuje z uradno spletno stranjo ali pa jo v celoti nadomesti.

7.1 Nadaljnje delo

Projekt kot tak še ni pripravljen na dokončno izdajo. Potrebno bi bilo poskrbeti za določene pomanjkljivosti in slabosti:

- **Varnost.** Za varnost je v sistemu nekoliko sicer poskrbljeno, a ne dovolj za uporabo v realnem podjetju. Vsi podatki, ki se prenašajo med Android aplikacijo in strežnikom, bi morali biti kriptirani. Prav tako bi bilo potrebno bolje urediti zaščito proti podvojevanju uporabnikov in izkoriščanju rezervacijskega sistema.

- **Enostavnost uporabe.** Tu govorimo predvsem o enostavnosti upravljanja s podatkovno bazo s strani administratorja. V trenutnem stanju strežnika se na določenih mestih od administratorja pričakuje, da pozna strukturo baze in oblikuje svoje podatke v skladu s pričakovano strukturo, saj drugače sistem javi napako oz. ne deluje kot bi moral. Predvsem gre za področje storitev in kateri od frizerjev določene storitve opravljajo. Tako, kot je zastavljeno trenutno, mora administrator za vsakega novega frizerja, ki ga doda, v tabeli 'Storitve' dodati nov atribut, ki predstavlja tega frizerja in mu določi dovoljene storitve.

- **Uporabniški vmesnik.** Vmesnik v Android aplikaciji je v trenutnem stanju precej osnoven. Potrebno bi ga bilo osvežiti in mu dodati določene funkcije, da bi deloval tako, kot se pričakuje danes od tovrstne aplikacije. Barvna shema bi se lahko samodejno prilagodila glede na temo, ki jo ima uporabnik izbrano v operacijskem sistemu. Tudi velikost pisave bi lahko bila odvisna od uporabnikovih želja.

- **Uporabniška izkušnja.** Za boljšo uporabniško izkušnjo bi bilo potrebno tudi urediti boljši sistem za opozarjanje strank v primeru, da je bila njihova zahteva za rezervacijo bodisi sprejeta ali zavrnjena. To bi opravili s preprostimi obvestili preko operacijskega sistema Android. Za to pa bi morali še omogočiti prijavo v aplikacijo z računom Google.

Tekom samega razvoja je prišlo na dan še veliko možnih izboljšav za trenutni sistem in veliko novih idej, ki bi lahko bile v prihodnosti implementirane. Prva stvar, ki bi bila potrebna, je dejanska spletna stran namenjena obiskovalcem salona, ki bi delovala podobno kot aplikacija Android in bi preko spletnega brskalnika opravljala podobno funkcijo. Sedanja administracija bi bila le del te strani. Dodali bi lahko seveda tudi iOS in Windows Phone aplikaciji.

Literatura

[1] Sun Microsystems. Dosegljivo:
https://www.oracle.com/sun/

[2] Programski jezik Java. Dosegljivo:
http://searchsoa.techtarget.com/definition/Java

[3] Programski jezik C#. Dosegljivo:
https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

[4] Označevalni jezik za oblikovanje večpredstavnostnih dokumentov. Dosegljivo:
http://www.yourhtmlsource.com/starthere/whatishtml.html

[5] Kaskadne slogovne podloge. Dosegljivo:
https://www.w3.org/Style/CSS/

[6] Skriptni jezik JavaScript. Dosegljivo:
http://javascript.about.com/od/reference/p/javascript.htm

[7] Razor. Dosegljivo:
http://www.w3schools.com/aspnet/webpages_razor.asp

[8] Razlika med REST in SOAP. Dosegljivo:
http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/

[9] Ogrodje Bootstrap. Dosegljivo:
http://getbootstrap.com/

[10] Postavitev elementov v ogrodju Bootstrap. Dosegljivo: http://bootstrap-sass.happyfuncorp.com/bootstrap-sass/layout/README.html

[11] Knjižnica jQuery. Dosegljivo: https://jquery.com/

[12] Wikipedia jQuery. Dosegljivo: https://en.wikipedia.org/wiki/JQuery

[13] Microsoftovo ogrodje .NET (Dot NET). Dosegljivo: https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.110).aspx

[14] Model-pogled-nadzornik. Dosegljivo: https://en.wikipedia.org/wiki/Model-view-controller

[15] ASP.NET MVC. Dosegljivo: http://www.asp.net/mvc

[16] Aplikacijski programski vmesnik. Dosegljivo: https://en.wikipedia.org/wiki/Application_programming_interface

[17] ASP.NET Web API. Dosegljivo: http://www.asp.net/web-api

[18] Ogrodje Entity. Dosegljivo: http://www.codeproject.com/Articles/363040/An-Introduction-to-Entity-Framework-for-Absolute-B

[19] LINQ. Dosegljivo: http://www.c-sharpcorner.com/blogs/what-is-linq1

[20] Integrirano razvojno okolje Visual Studio. Dosegljivo: https://msdn.microsoft.com/en-us/library/fx6bk1f4(v=vs.90).aspx

[21] Integrirano razvojno okolje Android Studio. Dosegljivo: https://developer.android.com/studio/intro/index.html

[22] Gradle. Dosegljivo:
https://en.wikipedia.org/wiki/Gradle

[23] Pristopi pri delu z ogrodjem Entity. Dosegljivo:
http://www.geekytidbits.com/entity-framework-tech-talk/

[24] ASP.NET Scaffolding. Dosegljivo:
http://www.asp.net/visual-studio/overview/2013/aspnet-scaffolding-overview